**eSPI PRU-ICSS Firmware User Guide**

Applies to Product Release: 01.00.00.00

Publication Date: December 4, 2018

**Document License**

This work is licensed under the Creative Commons Attribution-NoDerivs 3.0 Unported License. To view a copy of this license, visit http://creativecommons.org/licenses/by-nd/3.0/ or send a letter to Creative Commons, 171 Second Street, Suite 300, San Francisco, California, 94105, USA.

**Contributors to this document**

Copyright (C) 2018 Texas Instruments Incorporated - http://www.ti.com/



Texas Instruments, Incorporated  
20450 Century Boulevard  
Germantown, MD 20874 USA

Texas Instruments, Incorporated  
20450 Century Boulevard  
Germantown, MD 20874 USA

**This document is intended to instruct users on the build and install process of the ICSS implementation of eSPI, and basic instruction on eSPI applications.**

**TABLE OF CONTENTS**

1 Requirements 4

2 Build/Load 4

2.1 Driver/Firmware 4

2.2 Application 4

2.2.1 To create the application CCS project, open a CMD window and enter the following: 4

2.2.2 To build the CCS application: 4

2.3 To load/run the binary: 5

3 eSPI Applications 5

4 Limitations 6

4.1 Dual and Quad mode not tested 6

4.2 Supported Commands 6

4.3 Single-depth Queues 6

4.4 No Response Modifier Support 6

4.5 I/O[1] Alert Not Supported 6

5 Known Issues 6

5.1 Reset Interrupts 6

5.2 ESPI Close 7

# Requirements

* Windows 7
* Processor SDK v5.2 for AM335x
* Code Composer Studio v8.1.0.00011
* Beagle Bone Black rev. C

# Build/Load

*These instructions assume Processor SDK v5 is already installed on Windows 7*

<PDK> = **C:\ti\pdk\_am335x\_1\_0\_12\**

## Driver/Firmware

1. Copy the SPI package into **<PDK>\packages\drv\spi\**
2. To build the driver and firmware, open a CMD window and enter the following:
   1. **cd <PDK>\packages\**
   2. **pdksetupenv.bat**
   3. **gmake spi**
   4. **cd ti\drv\spi\**
   5. **gmake xdc\_meta**

## Application

### To create the application CCS project, open a CMD window and enter the following:

* 1. **cd <PDK>\packages\**
  2. **pdksetupenv.bat**
  3. **pdkProjectCreate.bat <SOC> all little spi**
     1. <SOC> = processor used (e.g. AM335x)
     2. This builds all SPI applications

### To build the CCS application:

* 1. Open CCS 8.1.0
  2. Create a new Workspace
  3. In the menu bar select **Project > Import CCS Projects…**
  4. Click **“Browse…”**
  5. Navigate to and select: **<PDK>\MyExampleProjects\**
  6. Select **“ESPI\_FwExample\_bbbAM335x\_armExampleProject”** from Discovered Projects
  7. From the Project Explorer, select **ESPI\_FwExample\_AM335x\_armExampleProject**
  8. Right-click the project name, and select **Build Project**

## To load/run the binary:

* 1. In the Target Configurations window, right click and select “New Target Configuration” and set up as follows:
     1. Connection: Select whichever debugger you have hooked up
     2. Board or device: **BeagleBone\_Black**
  2. Right-click your created CCXML from the Target Configurations window and select “Launch selected configuration”
  3. Menu-bar: **Run > Reset > System Reset**
  4. Right-click the CortexA8 core in the Debug window, select **“Connect Target”**
  5. Menu-bar: **Scripts > PRU\_ICSS > PRU\_ICSS\_Init**
  6. Menu-bar: **Run > Load > Load Program…** and navigate to: **<PDK>\packages\MyExampleProjects\ESPI\_FwExample\_bbbAM335x\_armExampleProject\Debug\ESPI\_FwExample\_bbbAM335x\_armExampleProject.out**
  7. Menu-bar: **Run > Resume** (if not already running automatically)
  8. Connect to PRU0
  9. **Run > Resume**
  10. Connect to PRU1
  11. **Run > Resume**

# eSPI Applications

We have included a simple example test application which handles PUT\_IORD\_SHORT, PUT\_IOWR\_SHORT, PUT\_MEMRD32\_SHORT, and PUT\_MEMWR32\_SHORT packets with expected address/data values. This can be used as a starting point to build your own applications, along with the following:

* eSPI uses the MCSPI API
  + Open as MCSPI instance 2
* HW interrupts must be enabled
  + i.e. HwiP\_enableInterrupt(espi\_cfg.intNum)
* MCSPI\_transfer() is blocking and takes a SPI\_Transaction parameter which is used as follows:
  + RX Transfer:
    - transaction.rxBuf = <some pre-allocated buffer>
    - transaction.txBuf = Null
    - transaction.count = max packet size
  + TX Transfer:
    - transaction.rxBuf = Null
    - transaction.txBuf = <some pre-allocated buffer>
    - transaction.count = <size of packet>
* The application must parse the full eSPI packet, and must respond with the full proper eSPI packet (the RSP code and CRC will be taken care of by the firmware)

# Limitations

## Dual and Quad mode not tested

eSPI supports Single, Dual, and Quad I/O modes, all are supported but only Single mode has been tested and proven to work at this point in time.

## Supported Commands

The following is a list of the supported eSPI commands, all other commands will result in NO\_RESONSE:

* GET\_STATUS
* GET\_CONFIGURATION
* SET\_CONFIGURATION
* PUT\_IORD\_SHORT (peripheral non-posted)
* PUT\_IOWR\_SHORT (peripheral non-posted)
* PUT\_MEMRD32\_SHORT (peripheral non-posted)
* PUT\_MEMWR32\_SHORT (peripheral posted)
* GET\_PC
* RESET

## Single-depth Queues

Only one packet per queue type can be handled. Currently supported queues are the Peripheral Non-Posted (NP) RX & TX queues, and the Peripheral Posted-Completion (PC) RX & TX queues. If the RX queue is full the corresponding bit will be updated in the Status Register, thus any attempt to put another packet in the same RX queue will result in a FATAL\_ERROR. Once the RX queue is cleared by the Host, any attempt to put another packet in the RX queue will result in NON\_FATAL\_ERROR, until the packet has been read from the corresponding TX queue.

## No Response Modifier Support

eSPI supports Response Modifier packets attached to the GET\_STATUS response, however we do not support them. Any SET\_CONFIGURATION requests to enable Response Modifiers will be ignored and not updated.

## I/O[1] Alert Not Supported

eSPI allows for either the I/O[1] line or the Alert# pin to signal an ALERT, however we only support signaling with the Alert# pin. This is NOT the default configuration, and the master needs to adjust accordingly by issuing a SET\_CONFIGURATION with the Alert Mode bit set to 1 in the General Capabilities and Configurations register.

# Known Issues

## Reset Interrupts

Currently Reset# interrupts are handled in the eSPI driver via GPIO, these will be moved to the application and the user will be responsible for handling a hard Reset.

## ESPI Close

The ESPI\_close\_v2() function isn’t properly implemented. This shouldn’t be too much of an issue for evaluation purposes, but keep in mind that closing an eSPI instance will leave HWI’s intact.